Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 19 (2013) 72 – 79

Procedia Computer Science

The 4th International Conference on Ambient Systems, Networks and Technologies (ANT 2013)

Gossiping Based Distributed Plan Monitoring[1]

Andrei Soeanu[a], Sujoy Ray[a], Mourad Debbabi[a], Mohamad Allouche[b], Jean Berger[b]

[a]*Concordia Institute for Information Systems Engineering, Concordia University, Montreal, Quebec, Canada*
[b]*Defence Research and Development Canada (DRDC) - Valcartier, Quebec, Canada*

Abstract

Joint plan execution is gaining momentum due to its benefits in terms of cost effectiveness and operational agility. In this paper, we introduce a lightweight gossip based multi-agent distributed protocol for plan execution monitoring in a dynamic environment characterized by unreliable communication links and exogenous events. The information obtained from the monitoring process can be used as support for detecting plan deviations and applying corrective measures. The contribution of this paper consists in the elaboration of an agent centric information sharing mechanism that is resilient to adverse changes in the execution environment by exhibiting a high degree of tolerance to communication errors. The distributed monitoring procedure is elaborated along with a relevant case study and experimental results.

Keywords: Plan Execution Monitoring, Multi-Agent Information Awareness, Gossiping Protocol, Resilience

1. Introduction and Background

Plan execution monitoring is essential to interpret the current state of the underlying work-flows. Monitoring focuses on variables related to the control functions [1] using well-chosen process parameters called global aggregates [2]. Classical monitoring aims toward a balanced trade-off between timely response and comprehensive data gathering. In this respect, multiple monitoring nodes report to a centralized coordination center [3, 4]. In contrast, distributed monitoring entails physically distributed and autonomous agents bound by local information that is shared in a dynamic and uncertain environment. Moreover, when the agents execution involves ambients, awareness is crucial in decision making for successful joint execution of common plans. In this respect, coordinating agents need to exchange data in order to stay informed about the events occurring outside their ambients. In the context of joint plan execution, complete joint information awareness requires complete knowledge of each agent of all the events generated by all agents. However, in practice the agents have less than complete knowledge when executing a plan in a dynamic environment characterized by unreliable communication. In this setting, we aim at achieving a high level of joint information awareness with respect to the generation of events within local agents' environment. Our

[1]This research is a result of a fruitful collaboration with Defence Research and Development Canada at Valcartier, Department of National Defense, MDA Corporation and Concordia University under the NSERC DND Research Partnership Program.

doi:10.1016/j.procs.2013.06.015

intent is to propose a general protocol that insures good information awareness, regardless of the structure of the agent organization. An agent oriented coalition formation depends on monitoring, organizational structure and resources for plan execution with strong motivation on information sharing for adaptive planning to ensure adequate response to changing circumstances. The potential changes in the communication and/or transport network conditions due to emergency situations, natural disasters, etc., are among the most challenging issues for the execution of operational and logistics plans both in civilian and military context.

At the national level, different agencies can inter-operate on various development or contingency plans while internationally, the partners of a multi-national coalition can share resources and information for increased cost effectiveness and agility during humanitarian aid distribution or peace keeping operations.

We introduce a distributed monitoring protocol that is lightweight and stateless. The plan execution is assumed to be taking place in a dynamic environment prone to the occurrence of disruptive exogenous events and unreliable communication. Thus, the objectives of this paper can be stated as a follows:

- Understand the limitation of current monitoring approaches relative to joint plan execution.
- Elaborate a distributed monitoring approach for achieving high level of joint information awareness.
- Propose a lightweight and effective agent centric monitoring protocol and related algorithms.
- Share important observations on the execution of the protocol and the related monitoring parameters.

In the context of plan execution monitoring, the usual setup assumes a central authority that holds the responsibility for the plan execution accomplishment. However, decentralization can bring specific benefits especially in the context of joint planning and execution monitoring. The need for decentralized execution (e.g. joint logistic support) stems from the present context of operation management in several sectors ranging from the provision of humanitarian aid to unit management in the theater of operations. In the foregoing context, an important aspect is the association of distributed nodes into clusters and cluster heads in order to localize the information exchange at the level of the distributed nodes while aggregating at the cluster head [5] the information relevant for the area covered by the cluster. Then, the cluster heads can similarly exchange relevant information for their area of influence. Figure 1 depicts a distributed node arrangement organized into clusters and cluster heads. In this setting, we specifically address the distributed monitoring problem involving autonomous agents participating in the joint execution of a generic plan (e.g. logistic delivery) in a dynamic environment with noisy communication links. During execution, various factors (e.g. environment) might prompt the agents to deviate from their established actions thus potentially requiring a re-planning. Thus, monitoring is necessary to correct the execution as needed. The aim is toward a lightweight, stateless distributed monitoring protocol whereby the agents can use simple rules and algorithms for effectively communicating in a disruptive environment. The proposed protocol allows to obtain and maintain a high level of shared information awareness of the overall situation. Moreover, in such distributed setting, there is no single point of failure and the agents can join or leave a cluster at any time.

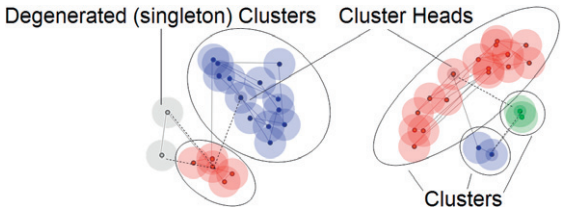

Fig. 1. Clustering technique for Join Plan Execution

The rest of the paper is organized as follows. Section 2 illustrates the state-of-the-art with respect to the relevant approaches for monitoring and associated challenges. It also identifies various shortcomings that are specific for different techniques within our scope of interest. Section 3 describes our approach along with the distributed monitoring protocol and the related algorithms. Section 4 presents the application of the approach on a case study followed by experimental results and analysis. Finally, Section 5 is summarizing our efforts and highlights the advantages, limitations and the scope of future work.

2. Related Work

Decentralization allows distributed self-organizing agents to pursue specific goals based on their abilities while aiming toward effective information dissemination. For governmental and military institutions, an established hierarchy forms a decision chain tree structure. Each level within the corresponding tree structure needs to receive information feeds from the level below in order to filter and extract the meaningful information to be sent to the level above. In [6] and [7], tree-based protocols are discussed where nodes organize themselves into a spanning tree. Each leaf node sends updates of its local variables to its respective parent. Each parent then computes its partial aggregate from the updates of its children. Hence, the aggregate computed at the root node represents the global aggregate. An up-to-date situational awareness requires that peers at the same level also exchange information. Several protocols for distributed computation of monitoring aggregates can be found in the literature [6, 8, 9, 10, 11]. Random-walk protocols [9] propagate node state information to a random neighbor, which updates and further relays the information to another randomly selected neighbor. Thus, the information gradually propagates to all nodes. Notwithstanding, an inherent issue to this procedure is the problem of scalable state representation. In [12], a gossip-based protocol supporting data dissemination through local communication is used to achieve reliable communication in dynamic network during emergency drill exercise. Asynchronous and round-based gossip (epidemic) aggregation protocols [10, 13] also exist for randomized communication. The core concept lies in the fact that each node holds an estimate of the aggregate that is typically converging exponentially to the true aggregate. In our scope of interest we are focusing on a solution with lightweight gossip-like capabilities in order to meet the requirements at the distributed peer level with the advantage of a reasonable information exchange rate without major scalability drawbacks. In this respect, we favor a clustering of peers whereby the peer-to-peer gossip interaction is limited to the neighborhood of a defined clustering distance [14] with small world structure. The definition of different clusters is the minimal requirement for the agent organization.

Alongside, correcting plan deviations represents a very important aspect in uncertain environments as exemplified by humanitarian aid distribution, rescue operations, disaster relief efforts or military crises. From an organizational perspective, at strategic level, the cost effectiveness represents an important concern. Moreover, decision makers may prefer a partial sharing of information with others as this may involve various policy or security issues. Therefore, distributed solution techniques may also include such aspects. In [15], an evolutionary learning approach is provided for near-optimally solving allocation and resource distribution problems in a collaborative setting with limited information sharing among participating peers.

3. Approach

In this paper, we assume for simplicity a decentralized setting whereby distributed agents run the same algorithms and procedures while maintaining full sharing of their information cooperatively. Nevertheless, this does not preclude partial information sharing if required for instance by some information exchange policy. In such case, the concept of joint information awareness is applicable for the information that can be shared. Given the nature of the plan execution environment in the context of our problem statement, the agents can benefit from techniques such as clustering and gossiping.

These techniques allow the agents to disseminate their updates in a manner that mitigates the effects of communication noise and uncertainty in the execution environment. Figure 2a. depicts the distributed agents setup. Assuming a similar event information value, gossiping is useful since it is characterized by periodic exchanges of short peer to peer messages. This provides resilience in case of message loss due to periodic retransmission. In a time unit, an agent can receive many messages from different peers but can disseminate a single one in the same time unit. The retransmission overhead might represent a concern but it can be addressed by maintaining a validity period (fresh information window) for the events to be communicated. However, along with the aforementioned fresh time window, an event priority queue is maintained based on which the agents select the events to be communicated. Also, whenever an agent is informed about an event that it is already aware, then the priority of the event will be lowered since the information is already disseminated to a certain degree and the priority should be on disseminating less known information. Moreover, if an event is getting out the fresh-window, it then represents old information

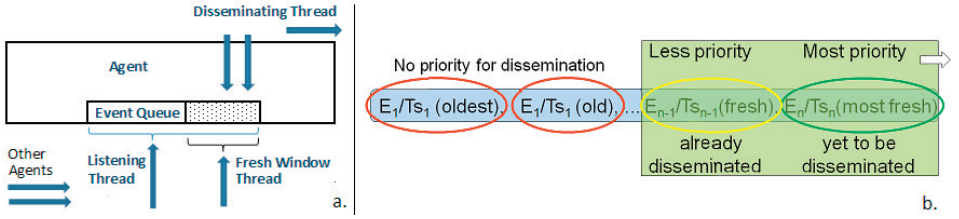

Fig. 2. Distributed Agent (a.) with Fresh-window for Event Dissemination (b.)

no longer relevant to be communicated and thus it will be discarded or rendered obsolete. Figure 2b. depicts the fresh-window usage. In our case, each communication initiating peer selects randomly another peer that is within an established clustering distance (or nearest peer if no peer is in clustering distance). Mobile agents can leave a cluster and join another. In the sequel, we consider the selection of the nearest peer when looking for peers within the clustering distance without mentioning it specifically.

The events are related to plan deviations such as change of plan variables (e.g. position relative to established routing path in the case of vehicle routing) or changes of status of different plan parameters (e.g. changes in the availability of different route segments for logistic delivery). We consider that with respect to the change of plan parameters (e.g. deviation in terms of position), there is a threshold that has to be crossed in order to generate an event such that the agents would not communicate insignificant changes. The joint information awareness level is the percentage of commonly known events (e.g. if agents A_1 and A_2 generate together a total of 100 events and A_1 knows 90 while A_2 knows 80, then the level of joint awareness is 80%).

We have a set of n distributed agents $D = \{A_1, A_2, \ldots, A_n\}$ and each agent A_k, $k \in \{1 \,..\, n\}$, is characterized by the tuple $\langle A_k^{actlist}, A_k^{act}, A_k^{dev} \rangle$ where:

- $A_k^{actlist}$ represents the list of actions (e.g. vehicle tour given as a route point sequence) assigned to A_k.
- A_k^{act} represents the current action (e.g. position change in terms of coordinates) of A_k.
- A_k^{dev} represents a scalar value for the absolute deviation of A_k.

The action list $A_k^{actlist}$ provides the plan variable changes (e.g. updated coordinates of the vehicle corresponding to the tour points) along with the start and end time to perform the variable changes (e.g. move from a tour point to the next one in sequence). In general, each agent A_k may use its own timer to determine its plan variable (e.g. position) updates from its assigned action list $A_k^{actlist}$. However, to make the approach more clear, we consider the use of a global clock Clk. Thus, at each time Clk_t, the expected variable changes of A_k is given by the function $CalcChg(A_k^{actlist}, Clk_t)$. The protocol involves an initiating phase where the agents are assigned their respective plan action lists (e.g. vehicle routes) and they establish their communication setup. Then the agents proceed to execute the plan by pursuing their respective action lists. Subsequently, during the plan execution, each agent A_k runs a thread $T^{dev}(A_k)$ that compares the plan variable values (e.g. position) of A_k to its currently assigned action list and calculates its absolute deviation A_k^{dev}. If the deviation exceeds a certain deviation threshold pTh, then the agent issues an event with the corresponding variable change (e.g. position) information and places it in its event queue along with the time-stamp of the event generation. In Figure 2, we can see the fresh-window thread that shifts the fresh-window over the event queue such that the events that exit the fresh-window are discarded or rendered obsolete but maintained for logging purposes. The figure also shows that each agent has a listening thread for receiving event notifications from other peer agents. Whenever such a notification is received the corresponding event and its time-stamp is placed in the agent event queue. In addition, each agent runs a disseminating thread that periodically checks the priorities of the events in the agent event queue and picks accordingly an event and selects a peer to transmit the event information and its time-stamp. The priority of an event is not strict, meaning that a higher priority event will just have a higher probability of being selected for dissemination whereas a lower priority event will have a lower probability of being selected. In this setting, when an agent is notified of an event that it already disseminated, it will lower its priority such

that it will have a lower probability of being selected for further dissemination. Thus, we have a weighting scheme for the probability of an agent selecting an event from the fresh-window. In order to assure that the most fresh events are disseminated faster, the weighting is following a normalized exponential decay, where the weight function terms have the form $w = 1/2^{\eta}$ with η representing the number of times that a disseminated event was notified back. The most fresh (yet to be disseminated) events are assigned by default to the pool with $\eta = 1$. A notified event that is already known and disseminated by an agent will move to a poll where the exponent values increases. Thus, if an event e is picked by agent A from the pool with $\eta = 1$ and disseminated to another peer and subsequently notified again, then e will move to the pool with exponent value $\eta' = \eta + 1 = 2$ with the corresponding weight of $1/2^2 = 0.25$. Through normalization, the gap to unity is distributed evenly among terms. For instance, if we have the terms 0.5 and 0.25, the gap to unity is 1-0.75=0.25 and after normalization, the terms are 0.625 and 0.375 assuring that their sum is unity. We employ the weighted die concept from [15] for the weighted pool selection and summarize it next in the context of our monitoring problem. A weighed die Δ is characterized by a weight vector $W = \{w_1, w_2, \ldots, w_M\}$, a repeatable "throw" action $t \in N^+$ and a set of possible pool selection outcomes $\{\delta_p \in \{0,1\} | p \in \{1..M\}\}$, where M is the number of pools to select from: $\Delta(W,t) \rightarrow \{\delta_1^t, \delta_2^t, \ldots, \delta_M^t\}$. In this setting, for a "throw" t, $\sum_{p=1}^{M} \delta_p^t = 1$ (only one outcome corresponds to each throw t) and for successive throws, we have $\{\forall p \in 1..M, \lim_{\tau\rightarrow\infty}(\sum_{t=1}^{\tau} \delta_p^t)/\tau = w_p\}$. Thus the die outcome distribution respects the weights in the vector W.

The proposed approach requires no direct acknowledgment mechanism and erroneous communication is discarded. Periodic retransmission takes place within the cluster such that the fresh information is quickly disseminated. Also, the reception of known information, serves as indirect communication feedback.

We provide at high level of abstraction the algorithms employed by each of the distributed agents. In essence, the core idea consists in the use of a tailored communication concept that is stateless (agents can join or leave a cluster at any point) and that allows the fresh information to be effectively disseminated with no direct acknowledgment. Furthermore, the employed procedure exhibits an adaptive level of redundancy which allows for a dissemination rate limitation feedback. Thus, Algorithm 1 initiates and terminates each agent and its executing threads. Algorithm 2 and Algorithm 3 provide respectively the logic for the fresh-window thread and the disseminating thread. Finlay, Algorithm 4 stands for the notification thread.

Algorithm 1 Distributed Gossiping Monitoring for Agent A_k

1: Initialize the FreshWindow Thread for A_k
2: Initialize the Disseminating Thread for A_k
3: Initialize the NotificationWait Thread for A_k
4: Initialize the deviation tracking thread $T^{dev}(A_k)$
5: Wait for monitoring activity termination condition
6: Send *stop_signal* to all threads
7: End

Algorithm 2 FreshWindow Thread algorithm for Agent A_k

1: *mode* ← 'refresh'
2: **while** *stop_signal* not received **do**
3: **if** *mode* = 'refresh' **then**
4: update the event queue of A_k with the notified or self generated events.
5: slide the fresh-window over the event queue to bring it up to date.
6: discarded or render obsolete the events with time-stamp outside the fresh-window.
7: *mode* ← 'wait'
8: **else**
9: wait for preemptive interruption or count the elapsing of an established refresh waiting interval;
10: *mode* ← 'refresh'
11: **end if**
12: **end while**

Algorithm 3 Disseminating Thread algorithm for Agent A_k

```
1: mode ← 'checking'
2: while stop_signal not received do
3:   if mode = 'checking' then
4:     select a pool p of A_k using the weighted die model Δ.
5:     choose nondeterministically an event e from the pool p
6:     randomly select non-originator peer A' in clustering distance and send e with its time-stamp to A'.
7:     mode ← 'wait'
8:   else
9:     wait for preemptive interruption or count the elapsing of an established check waiting interval;
10:    mode ← 'checking'
11:  end if
12: end while
```

Algorithm 4 NotificationWait Thread algorithm for Agent A_k

```
1: initialize notifMap as associative array
2: while stop_signal not received do
3:   wait for preemptive interruption or the notification of input event e
4:   if input event e notified then
5:     form hashKey value from event e information and the time-stamp of e
6:     update notifMap with event e information using hashKey slot.
7:   end if
8: end while
```

4. Results and Analysis

In Figure 3, we present a case study involving the distributed monitoring of a generic plan involving 3 pairs of agents. We show the performance (in terms of resilience to communication error rate) for increasingly higher fresh-window sizes (20, 30 and 40), different event generation rates (low - 30%, med. - 60% and high - 90%) and full range of communication error rate (0% up to 100%)[2]. All information awareness graphs have 3 regions corresponding to high information awareness (approximately 85% and higher shared information), medium awareness(approximately 50% to 85%) and low awareness(below 50%).

For a fresh-window size of 20, we can see in Figure 3a. (up) the joint information awareness graph for different event generation rates and the full range of communication error rate. For high information awareness, the resilience to error can be observed up to 40% error rate. In Figure 3a. (down) we see the corresponding average delay graph versus error rate which peaks between 30% to 50%, from under 1 time unit (low generation rate) to 2.5 (high generation rate). The delay is considered only for shared information[3]. We can also note that for high event generation rate, the performance is degraded since a node can send at most once per time unit but it can potentially receive multiple times per time unit. Likewise, Figure 3b. and 3c. show the information awareness and delay graphs for window sizes of 30 and 40. We note an increasing resilience to errors up to 50% and respectively 55%. This is accompanied by an increased average delay between 40% to 60% error rate, from 2 to over 3 time units and respectively between 50% to 70% error rate, from 2.5 to over 3.5 time units[4]. Similarly, we have a performance degradation for high event generation

[2]We exercised the protocol in a simulator that we implemented in Java. Event generation rates are relative to the simulation time unit. The time unit is discrete but averaging may lead to fractional values which have only a statistical significance used for comparison.

[3]The delay counts the simulation units from event generation to that of joint awareness. At high error rate, many events will exit the fresh-window before successful transmission such that in extreme case, at 100% error, no message is received and no delay is incurred.

[4]Depending on the plan, one can consider a trade-off between more awareness and longer delay or less awareness and shorter delay.

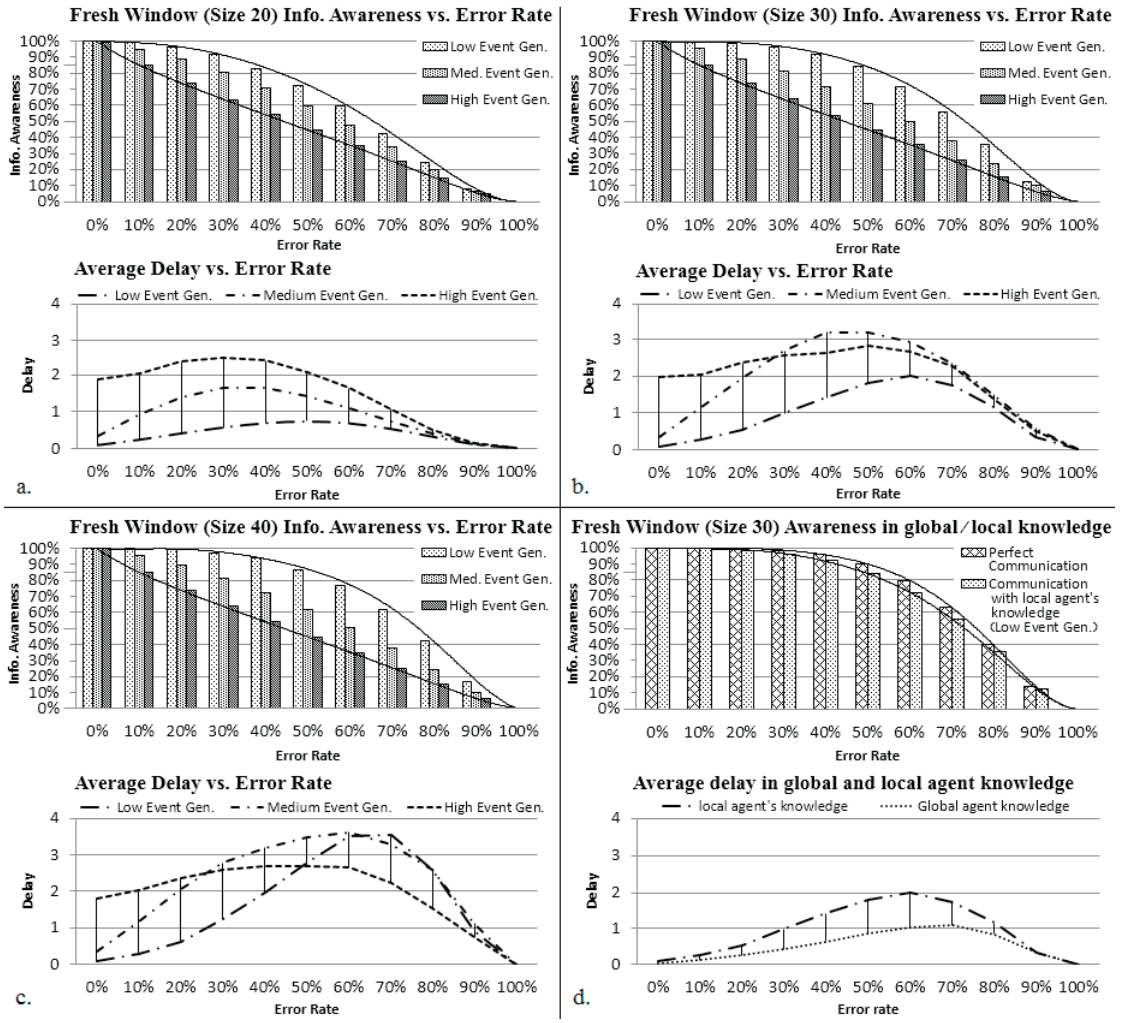

Fig. 3. Gossip Based Distributed Monitoring Case Study

rate. Thus, a lager fresh-window increases the performance for low generation rate since each agent is able to retransmit the fresh information to other peers before much more new information is generated in the cluster. Figure 3d. compares the results for a window size of 30 to those obtained in a similar case where the agents would have an idealized possibility to conduct perfect communication (no message would be sent to an agent already aware of it, which corresponds to ideal dissemination). We note a close performance for information awareness but a more notable difference for average delay (max. 2 units vs max. one unit).

In Figure 4, we show the results obtained for various cluster sizes, different event generation rates and increasingly higher error rates. We can note that overall, around 50% error rate can be tolerated for low event generation, around 30% error rate can be tolerated for medium event generation and around 10% error rate can be tolerated for high event generation. We can see highlighted the areas where the protocol performs well by providing high level of joint information awareness. Moreover, we can also expectedly note that for low event generation rate, the level of joint information awareness slowly decreases for larger cluster sizes while the delay slowly increases. For medium event generation rate, the dynamics is accentuating to some extent while for the case of high event generation rate, it accentuates more. However, as mentioned earlier, for high level of event generation rate, it is expected to have degraded performance since there is a limited opportunity to retransmit the fresh information.

	Cluster Size: 7			Cluster Size: 8			Cluster Size: 9			Cluster Size: 10		
Event rate	Error rate	Joint Info. awareness	Average delay	Error rate	Joint Info. awareness	Average delay	Error rate	Joint Info. awareness	Average delay	Error rate	Joint Info. awareness	Average delay
low	10.00%	99.07%	0.912	10.00%	98.69%	1.134	10.00%	98.07%	1.339	10.00%	97.24%	1.519
	30.00%	97.65%	1.786	30.00%	96.56%	2.095	30.00%	95.50%	2.343	30.00%	94.02%	2.587
	50.00%	91.94%	3.673	50.00%	89.32%	4.091	50.00%	86.29%	4.527	50.00%	83.81%	4.809
	70.00%	64.93%	6.990	70.00%	58.58%	7.867	70.00%	53.81%	8.675	70.00%	48.84%	9.293
med.	10.00%	97.75%	1.279	10.00%	96.66%	1.592	10.00%	95.38%	1.865	10.00%	93.92%	2.133
	30.00%	93.79%	2.851	30.00%	91.74%	3.267	30.00%	89.13%	3.748	30.00%	86.86%	4.024
	50.00%	82.85%	6.084	50.00%	78.90%	6.945	50.00%	74.20%	7.724	50.00%	70.55%	8.237
	70.00%	53.79%	11.377	70.00%	48.28%	12.687	70.00%	42.57%	14.039	70.00%	37.50%	15.241
high	10.00%	93.40%	1.820	10.00%	90.41%	2.257	10.00%	87.60%	2.673	10.00%	84.62%	3.068
	30.00%	84.74%	4.149	30.00%	80.24%	4.804	30.00%	76.12%	5.397	30.00%	71.55%	5.948
	50.00%	67.83%	8.377	50.00%	62.48%	9.458	50.00%	57.02%	10.587	50.00%	52.28%	11.389
	70.00%	40.06%	14.535	70.00%	34.83%	16.331	70.00%	30.14%	17.917	70.00%	25.47%	19.632

Fig. 4. Experimental Results

5. Conclusion and Future Work

In this paper, we presented a new stateless, agent centric peer-to-peer gossip protocol suitable for achieving high level of information awareness in execution environments with noisy communication. An exercise of the protocol was presented in a case study along with experimental results. Our findings show that the protocol exhibits a notable resilience to high levels of communication errors rates as long as there is enough retransmission opportunity relative to the overall event generation. The resilience represents an instrumental aspect in supporting distributed monitoring along with the ability of the agents to join or leave a cluster at any time due to the stateless nature of the protocol. Since we can mainly derive statistical guarantees for message delivery and delay values, this is a potential limitation, which will be addressed in future work by a further analysis of the protocol dynamics. Another future direction consists in extending the protocol with a tree-based structure to support information dissemination according to an established chain of command.

References

[1] F. Z. Wuhib, Distributed monitoring and resource management for large cloud environments, Ph.D. thesis, KTH- Royal Institute of Technology, Stockholm, Sweden (2010).

[2] F. Wuhib, R. Stadler, M. Spreitzer, Gossip-based resource management for cloud environments, in: CNSM, 2010, pp. 1–8.

[3] N. Delgado, A. Q. Gates, S. Roach, A taxonomy and catalog of runtime software-fault monitoring tools, IEEE Transactions on Software Engineering 30 (12) (2004) 859–872.

[4] K. L. Myers, Cpef: A continuous planning and execution framework, AI Magazine 20 (4) (1999) 63–69.

[5] A. K. Jain, R. C. Dubes, Algorithms for clustering data, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988.

[6] M. Dam, R. Stadler, A generic protocol for network state aggregation, in: In Proc. Radiovetenskap och Kommunikation (RVK, 2005, pp. 14–16.

[7] S. Madden, M. J. Franklin, J. M. Hellerstein, W. Hong, Tag: a tiny aggregation service for ad-hoc sensor networks, in: IN OSDI, 2002.

[8] N. Ahmed, D. Hadaller, S. Keshav, Incremental maintenance of global aggregates, Tech. Report CS-2006-19 (2006).

[9] C. Avin, C. Brito, Efficient and robust query processing in dynamic environments using random walk techniques, in: Proceedings of the 3rd international symposium on Information processing in sensor networks, IPSN '04, ACM, New York, NY, USA, 2004.

[10] M. Jelasity, A. Montresor, O. Babaoglu, Gossip-based aggregation in large dynamic networks, ACM Trans. Comput. Syst. 23 (2005) 219–252.

[11] R. V. Renesse, K. P. Birman, W. Vogels, Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining, ACM Transactions on Computer Systems 21 (2003) 164–206.

[12] O. Chipara, W. G. Griswold, A. N. Plymoth, R. Huang, F. Liu, P. Johansson, R. Rao, T. Chan, C. Buono, Wiisard: a measurement study of network properties and protocol reliability during an emergency response, in: Proceedings of the 10th international conference on Mobile systems, applications, and services, MobiSys '12, ACM, 2012, pp. 407–420.

[13] D. Mosk-Aoyama, D. Shah, Computing separable functions via gossip, in: Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing, PODC '06, ACM, New York, NY, USA, 2006, pp. 113–122.

[14] J. A. García, J. Fdez-Valdivia, F. J. Cortijo, R. Molina, A dynamic approach for clustering data, Signal Process. 44 (2).

[15] A. Soeanu, S. Ray, M. Debbabi, J. Berger, A. Boukhtouta, A learning based evolutionary algorithm for distributed multi-depot vrp, in: Proceeding of the 16th Int. Conf. on Knowledge-Based and Intelligent Information & Engineering Systems (KES), 2012.